UWB Qplatform API

Qorvo

Release R12.7.0-405-gb33c5c427

Contents

1 Overview

Qplatform is the set of platform specific services and drivers offered by a given Hardware platform.

Eventually, Qplatform will replace qhal, and aims at being the sum of different parts:

- UWB-Support Package: The fixed/known support layer for the UWB.
- BLE-Support Package : The fixed/known support layer for BLE/Matter.
- System Support Package : The fixed/known support layer the system (UWB, BLE or UWB + BLE).
- Drivers: Set of drivers offered by the platform. These drivers are not necessarily using a completely fixed API but they should offer at least a common minimal API (see work done on the QHAL front from BLE/Matter) and optionally some custom enhanced APIS. The drivers can be identified as being mostly generic or being completely platform specific. NB: The generic driver can be built on top of the specific ones to offer a simplified API.

On top of Qplatform, qosal offers OS level services (threads, mutexes, etc.).

To port the UWB stack on a given platform, the integrator must provide the UWB-Support package and System Support package.

The drivers are used:

- By the support package to fulfill the services;
- By the application itself when it makes sense.

1.1 Porting guide

1.1.1 Initialization

API *qplatform_init()* allows to initialize the platform system.

It aims at configuring everything which is specific to the platform. For example, it can include (but is not restricted to):

- GPIO configuration
- Powering up/down some peripherals, and initializing them
- Uploading some firmwares
- Etc.

1.1.1.1 QM33 requirements

QM33/DW3000 is a UWB transceiver-only chip. The UWB stack always run on an external MCU, and uses `dwt_uwb_driver` to communicate with the transceiver, using a SPI bus. The platform initialization requires to **configure the SPI driver**. Besides, multiple components (qhal_qotp, l1) of the UWB stack requires the `dwt_uwb_driver` **driver to be probed**.

qplatform_init() is responsible of both SPI and `dwt_uwb_driver` initializations.

Two Qplatform implementations are provided for QM33:

- The first implementation aims at being used with **Zephyr OS**. It can be built by enabling `CONFIG_QPLATFORM_IMPL_QM33_QHAL_ZEPHYR` and relies on Zephyr device tree.
- The second implementation aims at being used for non-zephyr nRFx platforms. It can be built by enabling `CONFIG_QPLATFORM_IMPL_QM33_QHAL_NON_ZEPHYR` and relies on Kconfig parameters to define the platform GPIOs and SPI instance:
  - `CONFIG_DWT_RSTN_GPIO_PORT` and `CONFIG_DWT_RSTN_GPIO_PIN` for **RSTn** GPIO;
  - `CONFIG_DWT_IRQ_GPIO_PORT` and `CONFIG_DWT_IRQ_GPIO_PIN` for **IRQ** GPIO;
  - `CONFIG_SPI_UWB_SCK_GPIO_PORT` and `CONFIG_SPI_UWB_SCK_GPIO_PIN` for **SPI CLK** GPIO;
  - `CONFIG_SPI_UWB_MOSI_GPIO_PORT` and `CONFIG_SPI_UWB_MOSI_GPIO_PIN` for **SPI MOSI** GPIO;
  - `CONFIG_SPI_UWB_MISO_GPIO_PORT` and `CONFIG_SPI_UWB_MISO_GPIO_PIN` for **SPI MISO** GPIO;
  - `CONFIG_SPI_UWB_CS_GPIO_PORT` and `CONFIG_SPI_UWB_CS_GPIO_PIN` for **SPI CS** GPIO;
  - `CONFIG_UWB_SPI_INSTANCE` for the **SPI instance** to use.

Important: Both implementation relies on qhal_qspi from qhal, and requires that module to be ported on the platform.

Note: `qspi` is planned to be moved from qhal to Qplatform in the future.

2 Qplatform API

2.1 qplatform_init

enum qerr **`qplatform_init`**(void)

Initialize the platform.

Parameters

- **`void`** – no arguments

2.1.1 Description

Initialize what is platform specific in the system. It should be called prior to any other init of the UWB stack.

2.1.2 Return

QERR_SUCCESS or error.

2.2 qplatform_deinit

enum qerr **`qplatform_deinit`**(void)

Denitialize the platform.

Parameters

- **`void`** – no arguments

2.2.1 Description

Deinitialize what is platform specific in the system.

2.2.2 Return

QERR_SUCCESS or error.

2.3 qplatform_get_wakeup_latency

enum qerr **`qplatform_get_wakeup_latency`**(uint16_t *wakeup_latency_us)

Get the wake-up latency, including both UWB and MCU latencies.

Parameters

- **`wakeup_latency_us`** (`uint16_t*`) – the returned wake-up latency, in microseconds.

2.3.1 Return

QERR_SUCCESS or error.

2.4 qplatform_uwb_interrupt_enable

enum qerr **qplatform_uwb_interrupt_enable**(void)

Enable interrupts for the UWB subsystem.

Parameters

- **void** – no arguments

2.4.1 Return

QERR_SUCCESS or error.

2.5 qplatform_uwb_interrupt_disable

enum qerr **qplatform_uwb_interrupt_disable**(void)

Disable interrupts for the UWB subsystem.

Parameters

- **void** – no arguments

2.5.1 Return

QERR_SUCCESS or error.

2.6 qplatform_uwb_spi_set_fast_rate_freq

void **qplatform_uwb_spi_set_fast_rate_freq**(void)

Configure fast rate frequency for SPI used for the UWB communication, if applicable.

Parameters

- **void** – no arguments

2.7 qplatform_uwb_spi_set_slow_rate_freq

void **qplatform_uwb_spi_set_slow_rate_freq**(void)

Configure slow rate frequency for SPI used for the UWB communication, if applicable.

Parameters

- **void** – no arguments

2.8 qplatform_uwb_reset

void **qplatform_uwb_reset**(void)

Performs UWB transceiver pin reset.

Parameters

- **void** – no arguments

2.9 qplatform_get_idle_timer_config

void **qplatform_get_idle_timer_config**(struct qtimer_config const **config)

Get the configuration of the idle timer.

Parameters

- **config** (`struct qtimer_config const**`) – Configuration of the idle timer.

2.10 qplatform_get_idle_timer_instance

uint8_t **qplatform_get_idle_timer_instance**(void)

Get the instance of the idle timer.

Parameters

- **void** – no arguments

2.10.1 Return

The instance of the idle timer.

Index